Universiteti i Prishtinës "Hasan Prishtina" Kosovë

Programimi në www

Pjesa 11 – JS (3)

Prof. Asoc. Dr. Ermir Rogova

Lecture content

- In this lecture we will cover the following:
  - Arrays
  - Events
  - Errors

Intro to arrays

- JavaScript arrays are used to store multiple values in a single variable.

```
var cars = ["Saab", "Volvo", "BMW"];
```

- Spaces and line breaks are not important. A declaration can span multiple lines:

```
var cars = [
  "Saab",
  "Volvo",
  "BMW"
];
```

Intro to arrays

- The following example also creates an Array, and assigns values to it:

```
var cars = new Array("Saab", "Volvo", "BMW");
```

- The two examples above do exactly the same.
- There is no need to use new Array().
- For simplicity, readability and execution speed, use the previous example (the array literal method).

Access the Elements of an Array

- You access an array element by referring to the **index number**.

- This statement accesses the value of the first element in cars:

```
var name = cars[0];
```

Example

```
var cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars[0];
```

- **Note:** Array indexes start with 0.
- [0] is the first element. [1] is the second element, etc…

Changing an Array Element

- This statement changes the value of the first element in cars:

```
var cars = ["Saab", "Volvo", "BMW"];
cars[0] = "Opel";
document.getElementById("demo").innerHTML = cars[0];
```

Access the Full Array

- With JavaScript, the full array can be accessed by referring to the array name:

```
var cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars;
```

The length Property

- The length property of an array returns the length of an array (the number of array elements).

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.length;   // the length of fruits is 4
```

- The length property is always one more than the highest array index.

The first and last elements

```
fruits = ["Banana", "Orange", "Apple", "Mango"];
var first = fruits[0];
```

```
fruits = ["Banana", "Orange", "Apple", "Mango"];
var last = fruits[fruits.length - 1];
```

Looping Array Elements

- The safest way to loop through an array, is using a for loop

```
var fruits, text, fLen, i;
fruits = ["Banana", "Orange", "Apple", "Mango"];
fLen = fruits.length;

text = "<ul>";
for (i = 0; i < fLen; i++) {
  text += "<li>" + fruits[i] + "</li>";
}
text += "</ul>";
```

- You can also use the Array.forEach() function

Array.forEach()

```
var fruits, text;
fruits = ["Banana", "Orange", "Apple", "Mango"];

text = "<ul>";
fruits.forEach(myFunction);
text += "</ul>";

function myFunction(value) {
  text += "<li>" + value + "</li>";
}
```

Adding Array Elements

- The easiest way to add a new element to an array is using the push() method:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.push("Lemon");    // adds a new element (Lemon) to fruits
```

- New element can also be added to an array using the length property:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits[fruits.length] = "Lemon";    // adds Lemon to fruits
```

Array holes

- Adding elements with high indexes can create undefined "holes" in an array:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits[6] = "Lemon";    // adds a new element (Lemon) to fruits
```

Avoid new Array()

- There is no need to use the JavaScript's built-in array constructor new Array(). **Use [] instead.**

- These two different statements both create a new empty array named points:

```
var points = new Array();          // Bad
var points = [];           // Good
```

- These two different statements both create a new array containing 6 numbers:

```
var points = new Array(40, 100, 1, 5, 25, 10);          // Bad
var points = [40, 100, 1, 5, 25, 10];          // Good
```

Avoid new Array()

- The new keyword only complicates the code. It can also produce some unexpected results:

```
var points = new Array(40, 100);  // Creates an array with two elements (40 and 100)
```

- What if I remove one of the elements?

```
var points = new Array(40);  // Creates an array with 40 undefined elements !!!!!
```

Converting Arrays to Strings

- The JavaScript method toString() converts an array to a string of (comma separated) array values.

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.toString();
```

- The join() method also joins all array elements into a string.
- It behaves just like toString(), but in addition you can specify the separator:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.join(" * ");
```

Popping and Pushing

- When you work with arrays, it is easy to remove elements and add new elements.
- This is what popping and pushing is:
- Popping items **out** of an array, or pushing items **into** an array.

Popping

- The pop() method removes the last element from an array:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.pop();           // Removes the last element ("Mango") from fruits
```

- The pop() method returns the value that was "popped out":

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
var x = fruits.pop();     // the value of x is "Mango"
```

Pushing

- The push() method adds a new element to an array (at the end):

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.push("Kiwi");      //  Adds a new element ("Kiwi") to fruits
```

- The push() method returns the new array length:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
var x = fruits.push("Kiwi");   //  the value of x is 5
```

Shifting and unshifting

- Shifting is equivalent to popping, working on the first element instead of the last.
- The shift() method removes the first array element and "shifts" all other elements to a lower index.
- The shift() method returns the string that was "shifted out"
- The unshift() method adds a new element to an array (at the beginning), and "unshifts" older elements
- The unshift() method returns the new array length.

Changing Elements

- Array elements are accessed using their **index number**:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits[0] = "Kiwi";       // Changes the first element of fruits to "Kiwi"
```

- The length property provides an easy way to append a new element to an array:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits[fruits.length] = "Kiwi";       // Appends "Kiwi" to fruits
```

Deleting Elements

- In JavaScript elements can be deleted by using the JavaScript operator delete:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
delete fruits[0];          // Changes the first element in fruits to undefined
```

- Using **delete** may leave undefined holes in the array.
- Use pop() or shift() instead.

Splicing an Array

- The splice() method can be used to add new items to an array:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(2, 0, "Lemon", "Kiwi");
```

- The first parameter (2) defines the position **where** new elements should be **added** (spliced in).
- The second parameter (0) defines **how many** elements should be **removed**.
- The rest of the parameters ("Lemon" , "Kiwi") define the new elements to be **added**.
- The splice() method returns an array with the deleted items

Merging (Concatenating) Arrays

- The concat() method creates a new array by merging (concatenating) existing arrays:

```
var myGirls = ["Cecilie", "Lone"];
var myBoys = ["Emil", "Tobias", "Linus"];
var myChildren = myGirls.concat(myBoys);   // Concatenates (joins) myGirls and myBoys
```

- The concat() method does not change the existing arrays. It always returns a new array.

Slicing an Array

- The slice() method slices out a piece of an array into a new array.
- This example slices out a part of an array starting from array element 1 ("Orange"):

```
var fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
var citrus = fruits.slice(1);
```

- The slice() method creates a new array. It does not remove any elements from the source array.
- The slice() method can take two arguments like slice(1, 3).
- The method then selects elements from the start argument, and up to (but not including) the end argument.

Sorting and reversing an Array

- The sort() method sorts an array alphabetically:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.sort();       // Sorts the elements of fruits
```

- The reverse() method reverses the elements in an array.

- You can use it to sort an array in descending order:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.sort();       // First sort the elements of fruits
fruits.reverse();    // Then reverse the order of the elements
```

Numeric sort

- By default, the sort() function sorts values as **strings**.
- This works well for strings ("Apple" comes before "Banana").
- However, if numbers are sorted as strings, "25" is bigger than "100", because "2" is bigger than "1".
- Because of this, the sort() method will produce incorrect result when sorting numbers.
- You can fix this by providing a **compare function**

The compare function

- The purpose of the compare function is to define an alternative sort order.
- The compare function should return a negative, zero, or positive value, depending on the arguments:

```
function(a, b){return a - b}
```

- When the sort() function compares two values, it sends the values to the compare function, and sorts the values according to the returned (negative, zero, positive) value.
- If the result is negative a is sorted before b.
- If the result is positive b is sorted before a.
- If the result is 0 no changes are done with the sort order of the two values.

Reacting to Events

- A JavaScript can be executed when an event occurs, like when a user clicks on an HTML element.
- To execute code when a user clicks on an element, add JavaScript code to an HTML event attribute

  onclick=*JavaScript*
- Examples of HTML events:
  - When a user clicks the mouse
  - When a web page has loaded
  - When an image has been loaded
  - When the mouse moves over an element
  - When an input field is changed
  - When an HTML form is submitted
  - When a user strokes a key

Example of HTML event

- In this example, the content of the <h1> element is changed when a user clicks on it

```
<!DOCTYPE html>
<html>
<body>
<h1 onclick="this.innerHTML='Ooops!'">Click on this text!</h1>
</body>
</html>
```

Example of HTML event

- In this example, a function is called from the event handler

```
<!DOCTYPE html>
<html>
<head>
<script>
function changetext(id)
{
id.innerHTML="Ooops!";
}
</script>
</head>
<body>
<h1 onclick="changetext(this)">Click on this text!</h1>
</body>
</html>
```

HTML Event Attributes

- To assign events to HTML elements we can use event attributes.
- How to assign an onclick event to a button element?

```
<button onclick="displayDate()">Try it</button>
```

- In the example above, a function named *displayDate* will be executed when the button is clicked.

Assigning Events Using the HTML DOM

- When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page.
- The **HTML DOM** model is constructed as a tree of **Objects**
- With the HTML DOM, JavaScript can access all the elements of an HTML document.
- Assigning an onclick event to a button element

```
<script>
document.getElementById("myBtn").onclick=function(){displayDate()};
</script>
```

onload and onunload Events

- The onload and onunload events are triggered when the user enters or leaves the page.
- The onload event can be used to check the visitor's browser type and browser version, and load the proper version of the web page based on the information.
- The onload and onunload events can be used to deal with cookies.

```
<body onload="checkCookies()">
```

The onchange Event

- The onchange event are often used in combination with validation of input fields.
- Below is an example of how to use the onchange. The upperCase() function will be called when a user changes the content of an input field.

```
<input type="text" id="fname" onchange="upperCase()">
```

The oninput Event

- The oninput event is often used in combination with validation of input fields.
- Below is an example of how to use the oninput. The upperCase() function will be called when a user changes the content of an input field.

```
<input type="text" id="fname" oninput="upperCase()">
```

onmouseover and onmouseout Events

- The onmouseover and onmouseout events can be used to trigger a function when the user mouses over, or out of, an HTML element.

onmousedown, onmouseup and onclick Events

- The onmousedown, onmouseup, and onclick events are all parts of a mouse-click.
- First when a mouse-button is clicked, the onmousedown event is triggered,
- then, when the mouse-button is released, the onmouseup event is triggered,
- finally, when the mouse-click is completed, the onclick event is triggered.

Errors – they WILL happen

- When the JavaScript engine is executing JavaScript code, different errors can occur
- It can be syntax errors, typically coding errors or typos made by the programmer.
- It can be misspelled or missing features in the language (maybe due to browser differences).
- It can be errors due to wrong input, from a user, or from an Internet server.
- And, of course, it can be many other unforeseeable things.

Statements

- When an error occurs, when something goes wrong, the JavaScript engine will normally stop, and generate an error message.
- The technical term for this is: JavaScript will **throw** an error.
- The **try** statement lets you to test a block of code for errors.
- The **catch** statement lets you handle the error.
- The **throw** statement lets you create custom errors.

try and catch

- The **try** statement allows you to define a block of code to be tested for errors while it is being executed.
- The **catch** statement allows you to define a block of code to be executed, if an error occurs in the try block.
- The JavaScript statements **try** and **catch** come in pairs.
- Syntax:

```
try
  {
  //Run some code here
  }
catch(err)
  {
  //Handle errors here
  }
```

The Throw Statement

- The throw statement allows you to create a custom error.
- The correct technical term is to create or **throw an exception**.
- If you use the throw statement together with try and catch, you can control program flow and generate custom error messages.
- The exception can be a JavaScript String, a Number, a Boolean or an Object
- Syntax:

  throw *exception*

More???

- For more examples on events check out

http://www.w3schools.com/js/js_ex_dom.asp

Pyetje???